

# VALUE AUTOMATA WITH FILTERS\*

Michaël Cadilhac<sup>(A)</sup>      Andreas Krebs<sup>(A)</sup>  
 Nutan Limaye<sup>(B)</sup>

<sup>(A)</sup>WSI, Universität Tübingen   <{cadilhac,krebs}@informatik.uni-tuebingen.de>

<sup>(B)</sup>Indian Institute of Technology Bombay   <nutan@cse.iitb.ac.in>

## Abstract

*We propose to study value automata with filters, a natural generalization of regular cost automata to nondeterminism. Models such as weighted automata and Parikh automata appear naturally as specializations. Results on the expressiveness of this model offer a general understanding of the behavior of the models that arise as special cases. A landscape of such restrictions is drawn.*

## 1. Introduction

Through their characteristic functions, formal languages are naturally seen as giving weights in  $\{0, 1\}$  to words. The transparent correspondence between these two views is provided by the notion of *weighted automata*, a widely studied model firmly rooted in sound algebraic concepts (see [12, 6] for recent expositions). Weighted automata provide an elegant framework to capture some functions from words to a value set, and their frequent use in the modeling of the qualitative aspects of real-life systems [2, 5] is a witness of their richness. In its simpler form, a weighted automaton can be thought as an automaton where transitions bear integer weights; on reading a word, the weights are multiplied along the path, and the final values for all paths labeled by the word are summed. This model is thus in some sense “restricted” to update its single register by multiplying it by constants along the path, and summing its different possible values.

In an effort to develop a theory of cost functions with a wider spectrum of update mechanisms, Alur *et al.* [1] introduced *regular cost functions*, a highly parametrizable framework in which a *variant* of weighted automata functions arises as a particular case. In doing so, they also introduced a *deterministic* model of automata, *cost register automata* (CRA), and studied in which settings this model matches regular cost functions. Similarly to regular cost functions, CRA are defined with respect to an underlying algebraic structure and restrictions on the update functions, but in this context, there exists an instance of these parameters that makes CRA precisely equivalent to weighted automata.

---

\*Ongoing work.

<sup>(B)</sup>The work of this paper was done during a stay of the third author at the Universität Tübingen.

This correspondence between CRA and weighted automata is however not a straightforward matter of renaming: weighted automata are intrinsically nondeterministic machines, while CRA are defined as deterministic automata. With an appropriate generalization of CRA to nondeterminism, in which weighted automata would arise as an obvious special case, the aforementioned correspondence would thus express that this case is *determinizable*.

Hence, the starting point of the present research is to propose a nondeterministic generalization of CRA that encompasses the behavior of weighted automata in a natural fashion. By computing on several registers at once, CRA however provide more information than weighted automata, and simply aggregating these registers into a single value seems to be a loss. We thus propose to add a *filtering* step at the end of the computation: the vector of registers should lay in a prescribed set. This allows models such as reversal-bounded counter machines [9], Parikh automata [10], or finite automata over free groups [11] to naturally have a quantitative counterpart. The fact that the test is made *at the end* of the computation ensures that the models stay decidable

This framework is sufficiently well-behaved to allow for general results that echo those to be found within special cases. It also offers a unified view of the limits inherent to submodels, where questions such as the following can be asked: “which properties of the underlying algebraic, update, and filter parameters are required for closure under a certain operation to hold?”

## 2. Definitions and preliminaries

A *monoid* is a set equipped with a binary associative operation having an identity element. For  $(M, +)$  and  $(N, \times)$  two monoids, a morphism  $h: M \rightarrow N$  is a function such that  $h(1) = 1$  and  $h(ab) = h(a)h(b)$ , for  $a, b \in M$ . A *semiring* is a set  $K$  equipped with two binary operations  $+$  and  $\times$  such that  $(R, +)$  is a commutative monoid with identity 0,  $(R, \times)$  is a monoid with identity 1 and absorbing element 0, and  $\times$  distributes over  $+$ .

A *semiautomaton*  $A$  is a tuple  $(Q, \Sigma, \delta)$ , where  $Q$  is a set of states,  $\Sigma$  is an alphabet, and  $\delta \subseteq Q \times \Sigma \times Q$  is a set of transitions. For a word  $w$ , we write  $\text{Paths}(A, w) \subseteq \delta^*$  for the set of paths on  $A$  labeled  $w$ . For a path  $\pi$ , we write  $\text{Orig}(\pi)$ , resp.  $\text{Dest}(\pi)$ , for the state in which the path starts, resp. ends. The semiautomaton is said to be *unambiguous* if there does not exist two different paths with same origin, destination, and label.

A *weighted automaton*  $W$  over a semiring  $K = (K, +, \times)$  is a tuple  $(A, I, U, C)$  such that  $A = (Q, \Sigma, \delta)$  is a semiautomaton,  $U: \delta^* \rightarrow (K, \times)$  is a morphism, and  $I, C: Q \rightarrow K$ . For a word  $w \in \Sigma^*$ , the automaton computes the value  $W(w)$  defined by:

$$W(w) = \sum_{\pi \in \text{Paths}(A, w)} I(\text{Orig}(\pi)) \times U(\pi) \times C(\text{Dest}(\pi)) ,$$

where  $\sum$  is the iterated  $+$ . The class of functions computed by weighted automata on  $K$  is denoted  $\text{WA}(K)$ . Moreover,  $W$  is *unambiguous* if  $A$  is, and we denote  $\text{UnWA}(K)$  the class of

functions computed by unambiguous weighted automata on  $K$ —note that in such automata, the actual interpretation of  $+$  is not needed.

### 3. Value automata with filters

Informally, this model differs from a weighted automaton on a semiring  $(K, +, \times)$  in two aspects:

1. The weighted automaton updates *its only* register  $x$  with actions of the form  $x \leftarrow x \times c$ , where  $c$  is determined by the transition. In value automata, more than one register can appear, and the update expressions can be algebraic expressions.
2. If a word is recognized by  $n$  different paths, the values  $x^{(1)}, x^{(2)}, \dots, x^{(n)}$  of  $x$  at the end of each path are *aggregated* using  $+$ . In contrast, value automata with filters apply a sieve on the different values of  $x$ , aggregating only those belonging to a prescribed set.

With generalization in mind, we wish to present a formalization that does not impose any *a priori* structure on the objects at hand, namely the underlying algebraic structure, the update functions, and the filter sets.

#### 3.1. Algebraic structures and classes

**Definition 3.1** *An algebraic structure  $K$  is composed of a base set (written also  $K$ ) and multiple internal operations, one of which is a distinguished operation called the aggregate. We assume that this operation is commutative, can take an unbounded number of arguments (associativity implying such a property), and acts as the identity when applied on a single element. This very general concept allows  $K$  to be a monoid, a semiring, etc.*

Let  $K$  be an algebraic structure and  $f: K^m \rightarrow K^n$  be a function. We will always assume that the functions can be expressed using algebraic expressions, that is: each entry of  $f(\vec{x})$  can be defined using an algebraic expression relying on the operations of  $K$  and the variables in  $\vec{x}$ . This corresponds to the idea that  $f$  should be an update function, and should not, for instance, make tests if  $K$  is not itself equipped with a test mechanism. Further, we say that  $f$  is *copyless* if the set of expressions used to define all the entries of  $f(\vec{x})$  contains at most one occurrence of each component of  $\vec{x}$ . We say that  $f$  is *moveless* if for all  $i$ , the  $i$ -th component of  $f(\vec{x})$  is not influenced by the values of  $x_j$ ,  $j \neq i$ . Finally, we say that  $f$  is *resetless* if no component of  $f(\vec{x})$  is constant when  $\vec{x}$  varies.

**Definition 3.2** *An update class  $\text{Upd}$  on  $K$  is a function that maps each pair  $(m, n) \in \mathbb{N} \times \mathbb{N}$  to a set of functions from  $K^m$  to  $K^n$ . We add in superscript  $\text{cl}$ ,  $\text{ml}$ , or  $\text{rl}$  when we consider the copyless, moveless, or resetless subsets of  $\text{Upd}$  (e.g.,  $\text{Upd}^{\text{cl}, \text{ml}}$ ). In our presentation, update classes will serve as restrictions on the register updates.*

**Examples 1** • When no restriction is imposed, that is, when updates are of the form of any algebraic expressions on  $K$ , we denote the update class  $\top$ .

- Let  $+$  and  $\times$  be two distinguished operations of  $K$ . We let **Affine** be the update class of affine functions, i.e., every  $f \in \text{Affine}(m, n)$  is defined using a matrix  $K^{n \times m}$  and a vector  $\vec{v} \in K^n$  by  $f(\vec{x}) = M.\vec{x} + \vec{v}$ . This class is succinctly written “ $+, \times c$ ” in [1], as it corresponds to updates using additions and multiplications by constants.
- With  $K$  as previously, 0 and 1 respectively absorbing and neutral elements for  $\times$ , and  $m = n$ , restrain **Affine** by having  $M$  be a 0–1-matrix with exactly one 1 per row. We obtain the update class **Trans**, so denoted as updates are of the form  $x_i \leftarrow x_j + c$ . This class is written “ $+c$ ” in [1].
- Similarly, when restraining **Affine** by having  $\vec{v}$  be the null vector, and  $M$  have at most one nonzero entry per row, we obtain the update class **Scale**, corresponding to updates of the form  $x_i \leftarrow x_j \times c$ .

**Definition 3.3** A filter class **Filt** on  $K$  is a map  $d \in \mathbb{N} \mapsto 2^{K^d}$ .

**Examples 2** • The filter class  $\top$  maps  $d$  to  $K^d$ . As a filter, its action is then void.

- Let  $+$  be a distinguished operation of  $K$  and  $<$  an order on  $K$ . The  $\text{FO}[+]$ -definable sets are those expressible as a first-order formula using  $+$ ,  $<$ , and constants. More precisely,  $F \subseteq K^d$  is  $\text{FO}[+]$ -definable if there is such a formula  $\varphi$  with  $d$  free variables such that  $\vec{x} \in F$  iff  $K \models \varphi(\vec{x})$ . We write this filter class  $\text{FO}[+]$ . For  $K = \mathbb{N}, \mathbb{Z}$  this corresponds to the *semilinear sets*, that is, finite unions of sets of the form  $\vec{v} + K.\vec{v}' + K.\vec{v}'' + \dots$ , where the  $\vec{v}$ ’s are in  $K^d$  [8]. We write  $\emptyset[+]$  for the *quantifier-free* variant of this filter class, that is, sets corresponding to formulas with no quantifiers.

### 3.2. Value automata with filters

**Definition 3.4 (Value Automaton with Filter)** Let  $K$  be an algebraic structure, **Upd** an update class on  $K$ , and **Filter** a filter class on  $K$ . Write  $\oplus$  for the aggregate operation of  $K$ .

A Value Automaton with Filter (VAF) of dimension  $d$  is a tuple  $\mathcal{A} = (A, I, U, F, C)$  where:

- $A = (Q, \Sigma, \delta)$  is a semiautomaton,
- $I: Q \rightarrow K^d$  is a partial function called the initialization,
- $U: \delta \rightarrow \text{Upd}(d, d)$  is the update function,
- $F \subseteq K^d$  is a subset in  $\text{Filter}(d)$  called the filter.
- $C: Q \rightarrow \text{Upd}(d, 1)$  is a partial function called the collapse.

The VAF  $\mathcal{A}$  defines a partial function  $f: \Sigma^* \rightarrow K$  as follows. Intuitively, for a path in  $A$  labeled  $w$ , the registers are initialized with  $I(q)$ , for  $q$  the first state of the path. They are then updated using the functions of  $U$ , filtered by  $F$ , and merged into a single value by  $C(q')$ , for  $q'$  the last state of the path. All such values for a label  $w$  are then aggregated using the aggregate operation.

Formally, write  $U_t$  for  $U(t)$ , and  $C_q$  for  $C(q)$ . Define the valuation of a path  $\text{val}: \delta^+ \rightarrow K^d$  by  $\text{val}(t) = I(\text{Orig}(t))$ , for  $t \in \delta$ , and  $\text{val}(\pi t) = U_t(\text{val}(\pi))$ , with  $\pi \in \delta^*$ ,  $t \in \delta$ . In particular, if a

path starts in  $q$  and  $I(q)$  is undefined, then the valuation of the path is undefined. Finally:

$$f(w) = \bigoplus_{\substack{\pi \in \text{Paths}(A, w) \\ \text{val}(\pi) \in F}} C_{\text{Dest}(\pi)}(\text{val}(\pi)) ,$$

where we implicitly discard undefined values of  $\text{val}(\pi)$ , and the  $\oplus$  of zero elements is undefined.

The VAF is deterministic (*DetVAF*) if  $A$  is deterministic and  $I$  is defined on a single state. It is unambiguous (*UnVAF*) if  $A$  is. Finally, it is one-success (*OneVAF*) if  $|\{\pi \in \text{Paths}(A, w) \mid \text{val}(\pi) \in F\}| \leq 1$ , that is, if the aggregate is not needed.

The class of such automata is written  $\text{VAF}(K, \text{Upd}, \text{Filter})$ , and similarly for *DetVAF*, *UnVAF*, and *OneVAF*. We identify these classes with the classes of functions they define. For a distinguished element  $0$  of  $K$ , the  $0$ -support of a VAF is set of words it maps to  $0$ .

### 3.3. Models arising as special cases

**Cost register automata and weighted automata.** Deterministic VAF with  $\top$  as filter class are identical to the *cost register automata* of [1]. The expressiveness results therein relating different restrictions of cost register automata are summed up next:

**Theorem 3.5 ([1])** *With  $(K, +, \times)$  a semiring:*

1.  $\text{DetVAF}(K, \text{Scale}^{\text{cl}}, \top) \subsetneq \text{DetVAF}(K, \top^{\text{cl}}, \top) = \text{DetVAF}(K, \text{Scale}, \top) = \text{UnWA}(K) \subsetneq \text{DetVAF}(K, \top, \top),$
2.  $\text{DetVAF}(K, \text{Affine}^{\text{cl}}, \top) \subsetneq \text{DetVAF}(K, \text{Affine}, \top) = \text{WA}(K).$

Moreover, weighted automata are naturally expressed as nondeterministic VAF where only linear transformations are used—in which case, if no register moves are allowed and no filtering is made, having multiple registers does not increase the computing power. Hence, in light of the last point of Theorem 3.5:

**Proposition 3.6** *With  $(K, +, \times)$  a semiring,  $\text{VAF}(K, \text{Scale}^{\text{ml,rl}}, \top) = \text{DetVAF}(K, \text{Affine}, \top).$*

Weighted automata over *valuation monoids* constitute a recent fruitful effort towards generalizing weighted automata to less restricted settings [7]. The direction taken there is to move away from the iterative aspect of weighted automata. Indeed, a valuation monoid  $M$  is equipped with an extra function  $M^+ \rightarrow M$  intended to compute a value given the weights of each of the transitions in a path. Locality and incrementality being at the heart of our models, such automata do not seem to have an equivalent expression within VAF.

**Parikh automata and affine Parikh automata.** Parikh automata were introduced by Klaedkte and Rueß [10] and further studied and extended in [3]. They are defined as *recognizers*.

In a Parikh automaton, a set of (integer) counters is incremented during a run, and a word is accepted if the values belong to a prescribed semilinear set. An *affine Parikh automaton* is defined similarly, except that the update function is lifted to any affine transformation. Viewing languages as functions from  $\Sigma^*$  to  $\{0, 1\}$ , it is readily seen that:

**Proposition 3.7** *Deterministic, unambiguous, and nondeterministic Parikh automata (resp. affine Parikh automata) can be simulated by the corresponding sort of  $\text{VAF}(\mathbb{N}, \text{Trans}, \text{FO}[+])$  (resp.  $\text{VAF}(\mathbb{N}, \text{Affine}, \text{FO}[+])$ ).*

**Theorem 3.8 ([3, 4])** •  $\text{DetVAF}(\mathbb{N}, \text{Trans}^{\text{ml}}, \text{FO}[+])$   
 $\subsetneq \text{UnVAF}(\mathbb{N}, \text{Trans}^{\text{ml}}, \text{FO}[+]) = \text{DetVAF}(\mathbb{N}, \text{Trans}, \text{FO}[+]) = \text{UnVAF}(\mathbb{N}, \text{Trans}, \text{FO}[+])$   
 $\subsetneq \text{OneVAF}(\mathbb{N}, \text{Trans}^{\text{ml}}, \text{FO}[+]) \subseteq \text{VAF}(\mathbb{N}, \text{Trans}^{\text{ml}}, \text{FO}[+]) = \text{VAF}(\mathbb{N}, \text{Trans}, \text{FO}[+])$   
 •  $\text{DetVAF}(\mathbb{N}, \text{Affine}, \text{FO}[+]) = \text{UnVAF}(\mathbb{N}, \text{Affine}, \text{FO}[+]) \subsetneq \text{VAF}(\mathbb{N}, \text{Affine}, \text{FO}[+])$ ,  
 •  $\text{VAF}(\mathbb{N}, \text{Trans}, \text{FO}[+])$  and  $\text{DetVAF}(\mathbb{N}, \text{Affine}, \text{FO}[+])$  are incomparable.  
 • In these classes, substituting  $\mathbb{N}$  by  $\mathbb{Z}$  or  $\mathbb{Q}$  does not impact the the 0-support languages thus defined.

Finally, VAFs can keep an extra register to 1 and use the collapse function to return it; if the aggregate function of the algebraic structure is the sum, this counts the accepting paths:

**Proposition 3.9** *For any VAF, the function mapping a word to the number of paths whose valuations are in the filter is a function in the same class of VAFs. In particular, a VAF can compute the number of accepting paths in a Parikh automaton or an affine Parikh automaton.*

## 4. Conclusion

In this short exposition, we presented a new model that aims at identifying where some properties of models widely studied in the literature come from. In a longer version, we will present the closure properties, decidability properties, and expressiveness results that hold for the general setting. Our goal is then to identify the essential characteristics that falsify a given property. For instance, unambiguity adds no expressiveness as long as the set of functions  $x_i \leftarrow x_j$  is available as updates (as witnessed here by Theorem 3.8). Similarly, some separations between the deterministic and nondeterministic variants can be deduced from the outset, generalizing results for special cases. Another line of results is to work towards simplifying the filter set—for instance, using  $\emptyset[+]$  instead of  $\text{FO}[+]$  does not impact the expressiveness. Finally, for well-behaved algebraic structures, complexity results can be derived for the decidable problems.

## References

- [1] Rajeev Alur, Loris D’Antoni, Jyotirmoy Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *LICS*, pages 13–22. IEEE Computer Society, 2013.

- [2] Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Transactions on Algorithms*, 6(2):1–36, 2010.
- [3] Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Affine Parikh automata. *RAIRO - Theoretical Informatics and Applications*, 46(04):511–545, 2012.
- [4] Michaël Cadilhac, Alain Finkel, and Pierre Mckenzie. Unambiguous constrained automata. *Int. J. of Foundations of Computer Science*, 24(07):1099–1116, 2013.
- [5] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Transactions on Computational Logic*, 11(4):23, 2010.
- [6] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer Publishing Company, Incorporated, 1st edition, 2009.
- [7] Manfred Droste and Ingmar Meinecke. Weighted automata and regular expressions over valuation monoids. *Int. J. of Foundations of Computer Science*, 22(08):1829–1844, 2011.
- [8] Seymour Ginsburg and Edwin H. Spanier. Semigroups, Presburger formulas and languages. *Pacific J. Mathematics*, 16(2):285–296, 1966.
- [9] Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978.
- [10] Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In *ICALP*, volume 2719 of *LNCS*, pages 681–696. Springer-Verlag, 2003.
- [11] Victor Mitrana and Ralf Stiebe. Extended finite automata over groups. *Discrete Appl. Math.*, 108(3):287–300, 2001.
- [12] Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.